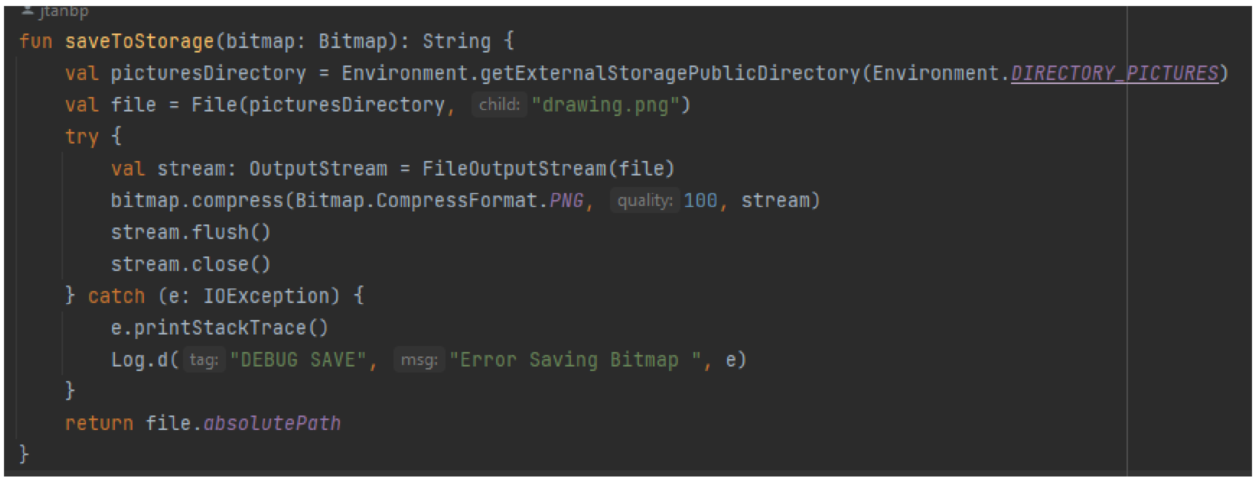
Phase 2 Code Review

Zidong Fan and Jowie Tan

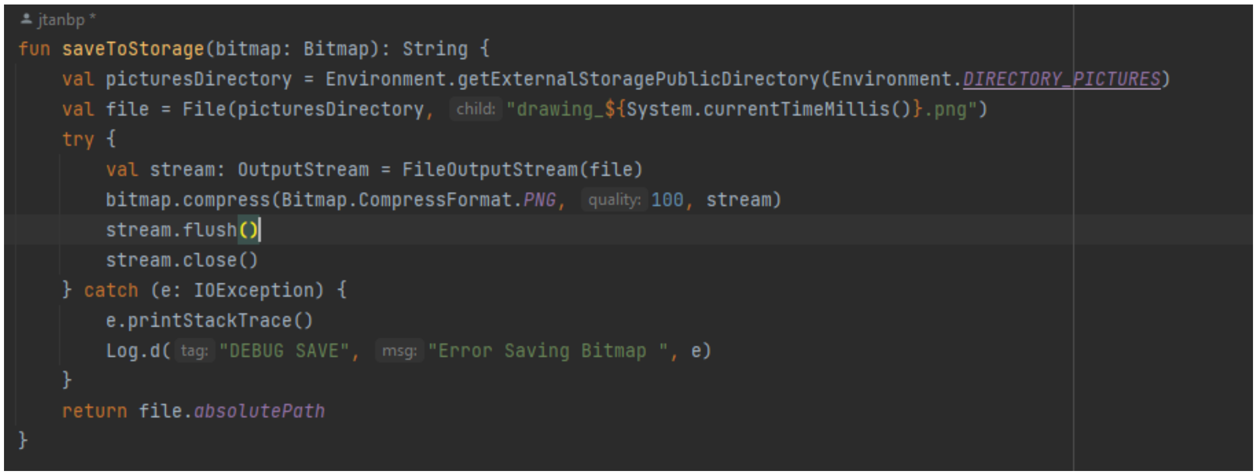
1. Improvements

Each person pick a piece of code from your own project (class, method, etc) that you think can be improved. Discuss that code in detail, improve it, and commit those improvements (ie, refactor something with a helper that has fresh eyes)

1.1. Jowie's Code

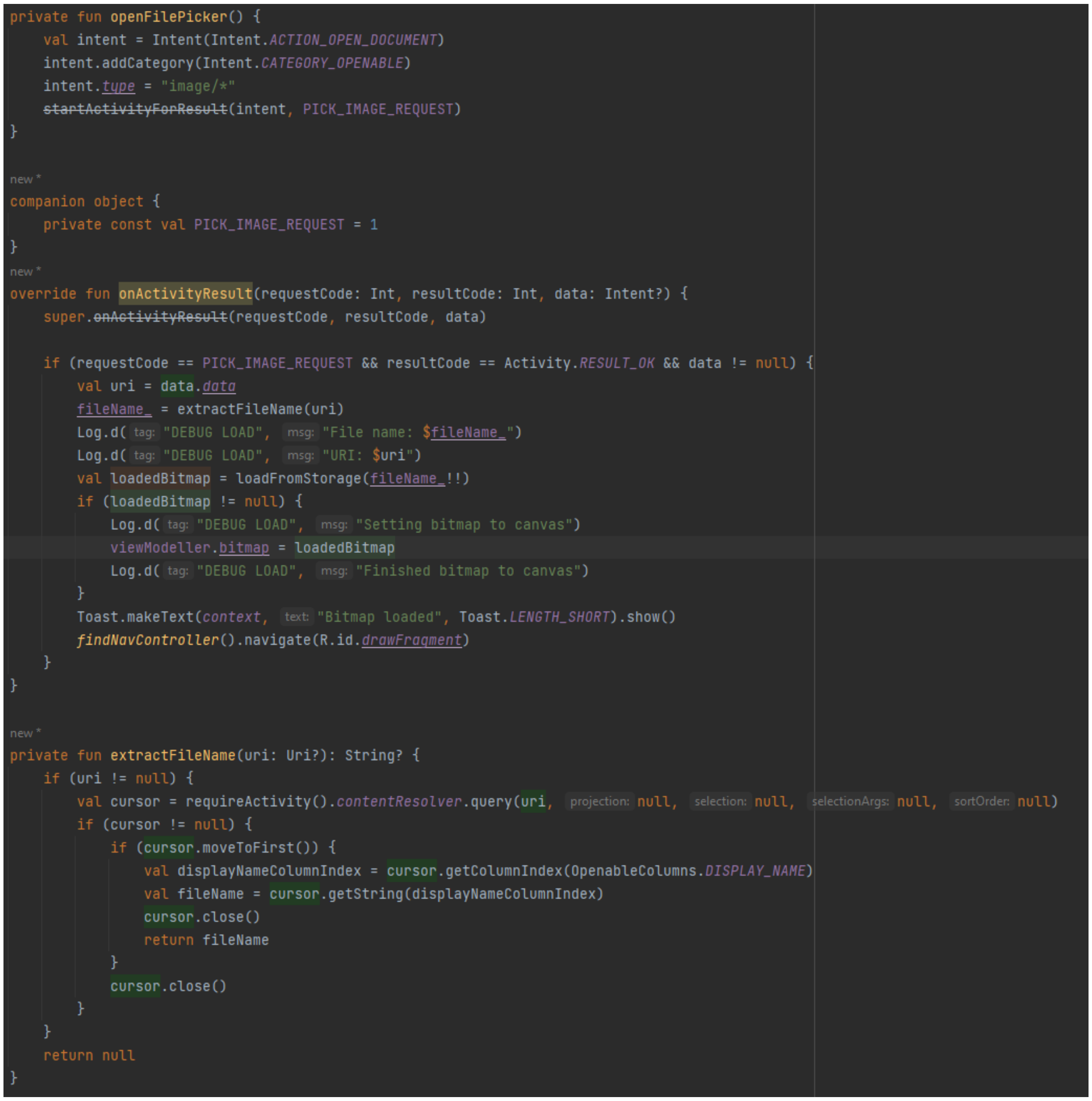


```
jtanbp
fun saveToStorage(bitmap: Bitmap): String {
    val picturesDirectory = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES)
    val file = File(picturesDirectory, child: "drawing.png")
    try {
        val stream: OutputStream = FileOutputStream(file)
        bitmap.compress(Bitmap.CompressFormat.PNG, quality: 100, stream)
        stream.flush()
        stream.close()
    } catch (e: IOException) {
        e.printStackTrace()
        Log.d( tag: "DEBUG SAVE", msg: "Error Saving Bitmap ", e)
    }
    return file.absolutePath
}
```



```
jtanbp *
fun saveToStorage(bitmap: Bitmap): String {
    val picturesDirectory = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES)
    val file = File(picturesDirectory, child: "drawing_${System.currentTimeMillis()}.png")
    try {
        val stream: OutputStream = FileOutputStream(file)
        bitmap.compress(Bitmap.CompressFormat.PNG, quality: 100, stream)
        stream.flush()
        stream.close()
    } catch (e: IOException) {
        e.printStackTrace()
        Log.d( tag: "DEBUG SAVE", msg: "Error Saving Bitmap ", e)
    }
    return file.absolutePath
}
```

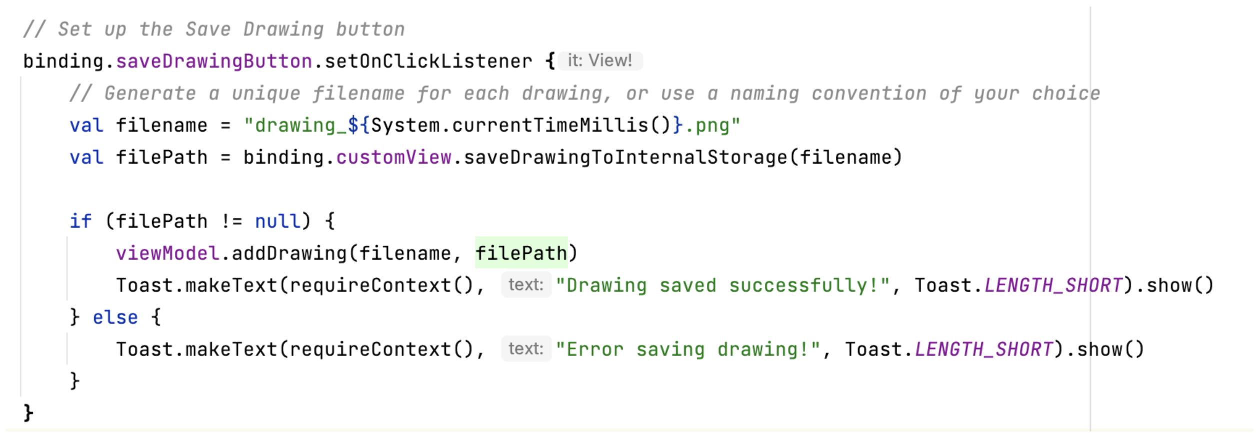
First, I had taken Zidong's suggestion to save drawings as a separate file based on the current timer.



```
private fun openFilePicker() {
    val intent = Intent(Intent.ACTION_OPEN_DOCUMENT)
    intent.addCategory(Intent.CATEGORY_OPENABLE)
    intent.type = "image/*"
    ~~startActivityForResult~~(intent, PICK_IMAGE_REQUEST)
}

new *
companion object {
    private const val PICK_IMAGE_REQUEST = 1
}
new *
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.~~onActivityResult~~(requestCode, resultCode, data)

    if (requestCode == PICK_IMAGE_REQUEST && resultCode == Activity.RESULT_OK && data != null) {
        val uri = data.data
        fileName_ = extractFileName(uri)
        Log.d( tag: "DEBUG LOAD",  msg: "File name: $fileName_")
        Log.d( tag: "DEBUG LOAD",  msg: "URI: $uri")
        val loadedBitmap = loadFromStorage(fileName_!!)
        if (loadedBitmap != null) {
            Log.d( tag: "DEBUG LOAD",  msg: "Setting bitmap to canvas")
            viewModeller.bitmap = loadedBitmap
            Log.d( tag: "DEBUG LOAD",  msg: "Finished bitmap to canvas")
        }
        Toast.makeText(context,  text: "Bitmap loaded", Toast.LENGTH_SHORT).show()
        findNavController().navigate(R.id.drawFragment)
    }
}


new *
private fun extractFileName(uri: Uri?): String? {
    if (uri != null) {
        val cursor = requireActivity().contentResolver.query(uri,  projection: null,  selection: null,  selectionArgs: null,  sortOrder: null)
        if (cursor != null) {
            if (cursor.moveToFirst()) {
                val displayNameColumnIndex = cursor.getColumnIndex(OpenableColumns.DISPLAY_NAME)
                val fileName = cursor.getString(displayNameColumnIndex)
                cursor.close()
                return fileName
            }
            cursor.close()
        }
    }
    return null
}
```

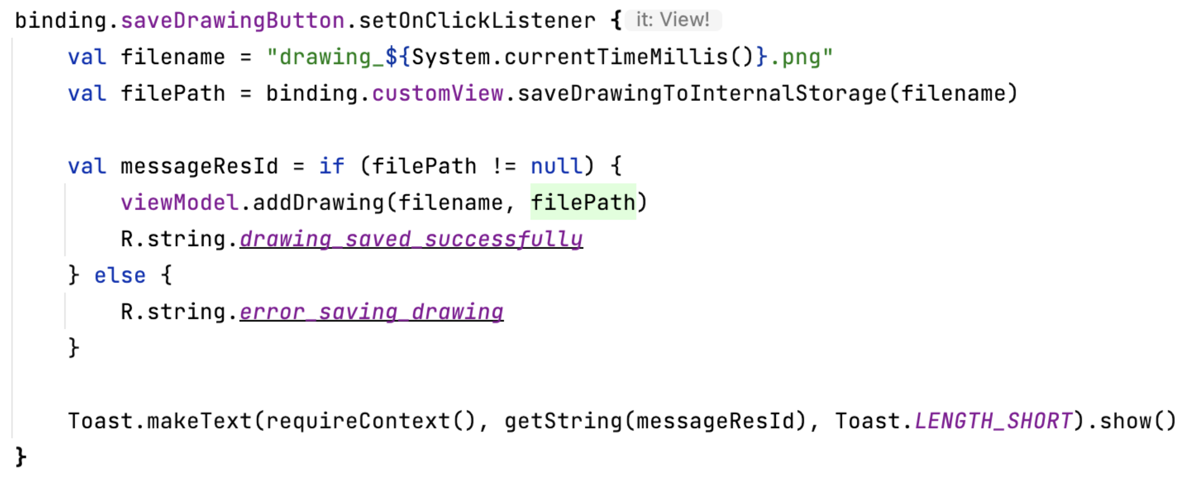
Second, in order to solve the problem of loading the file, I took awhile to write the code to open up a file explorer to navigate to the desired file to load. This way I could load the countless drawings saved.

1.2. Zidong’s Code



```kotlin
// Set up the Save Drawing button
binding.saveDrawingButton.setOnClickListener { it: View!
    // Generate a unique filename for each drawing, or use a naming convention of your choice
    val filename = "drawing_${System.currentTimeMillis()}.png"
    val filePath = binding.customView.saveDrawingToInternalStorage(filename)

    if (filePath != null) {
        viewModel.addDrawing(filename, filePath)
        Toast.makeText(requireContext(),  text: "Drawing saved successfully!", Toast.LENGTH_SHORT).show()
    } else {
        Toast.makeText(requireContext(),  text: "Error saving drawing!", Toast.LENGTH_SHORT).show()
    }
}
```

1. Used a resource string for the toast message to improve localization.
2. Utilized an if expression to determine the message resource ID based on the result of saving the drawing.
3. Used getString to retrieve the message from resources.



```kotlin
binding.saveDrawingButton.setOnClickListener { it: View!
    val filename = "drawing_${System.currentTimeMillis()}.png"
    val filePath = binding.customView.saveDrawingToInternalStorage(filename)

    val messageResId = if (filePath != null) {
        viewModel.addDrawing(filename, filePath)
        R.string.drawing_saved_successfully
    } else {
        R.string.error_saving_drawing
    }

    Toast.makeText(requireContext(), getString(messageResId), Toast.LENGTH_SHORT).show()
}
```

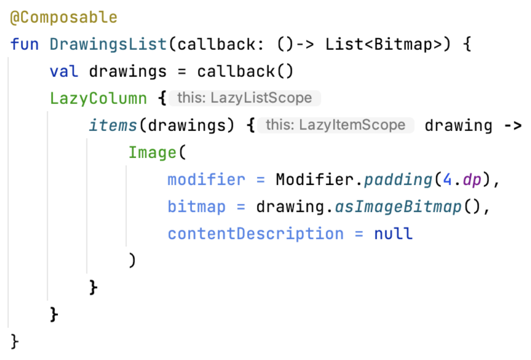
2. Similar Functionality

Pick a piece of functionality that both of your projects include. Discuss the similarities/differences between your implementations

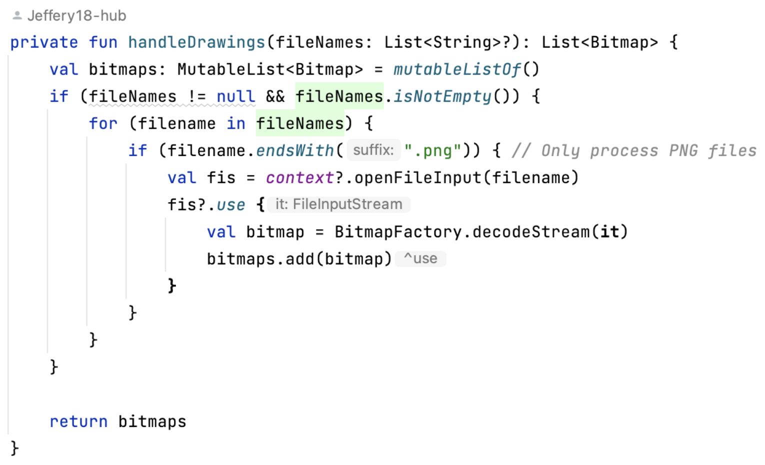
Jowie’s Code:

```
fun saveToStorage(bitmap: Bitmap): String {
    val picturesDirectory = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES)
    val file = File(picturesDirectory, "drawing.png")
    try {
        val stream: OutputStream = FileOutputStream(file)
        bitmap.compress(Bitmap.CompressFormat.PNG, 100, stream)
        stream.flush()
        stream.close()
    } catch (e: IOException) {
        e.printStackTrace()
        Log.d("DEBUG SAVE", "Error Saving Bitmap ", e)
    }
    return file.absolutePath
}

fun loadFromStorage(): Bitmap? {
    val picturesDirectory = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES)
    if (picturesDirectory == null) {
        Toast.makeText(context, "Failed to access storage", Toast.LENGTH_SHORT).show()
        return null
    }
    val file = File(picturesDirectory, "drawing.png")
    if (!file.exists()) {
        Toast.makeText(context, "No saved drawing found", Toast.LENGTH_SHORT).show()
        return null
    }
    val options = BitmapFactory.Options()
    options.inMutable = true
    return BitmapFactory.decodeFile(file.absolutePath, options)
}
```

Zidong's Code:



```
@Composable
fun DrawingsList(callback: ()-> List<Bitmap>) {
    val drawings = callback()
    LazyColumn { this: LazyListScope
        items(drawings) { this: LazyItemScope  drawing ->
            Image(
                modifier = Modifier.padding(4.dp),
                bitmap = drawing.asImageBitmap(),
                contentDescription = null
            )
        }
    }
}
```



```
Jeffery18-hub
private fun handleDrawings(fileNames: List<String>?): List<Bitmap> {
    val bitmaps: MutableList<Bitmap> = mutableListOf()
    if (fileNames != null && fileNames.isNotEmpty()) {
        for (filename in fileNames) {
            if (filename.endsWith( suffix: ".png")) { // Only process PNG files
                val fis = context?.openFileInput(filename)
                fis?.use { it: FileInputStream
                    val bitmap = BitmapFactory.decodeStream(it)
                    bitmaps.add(bitmap) ^use
                }
            }
        }
    }

    return bitmaps
}
```

We got 2 similar pieces of code that are similar in that they both involve loading images as Bitmaps from storage. However, they differ in terms of where and how they access and load the images.

1. Purpose:

- `loadFromStorage()`: This function is designed to load a specific image file named "drawing.png" from the public external storage directory.
- `handleDrawings()`: This function is designed to load multiple image files with names in a list from the internal app storage.

2. Storage Location:
- `loadFromStorage()`: It attempts to load an image from the public external storage directory using `Environment.getExternalStoragePublicDirectory()`.
- `handleDrawings()`: It loads images from the app's internal storage using `context?.openFileInput(filename)`.

3. Error Handling:
- `loadFromStorage()`: It displays Toast messages for error handling when it fails to access storage or when the specified file does not exist.
- `handleDrawings()`: It does not include error handling for cases where a file is not found or cannot be read.

4. File Format:
- `loadFromStorage()`: It specifically looks for a file named "drawing.png" in the Pictures directory.
- `handleDrawings()`: It processes all files in the provided list that have a ".png" extension.

5. Bitmap Loading:
- `loadFromStorage()`: It uses `BitmapFactory.decodeFile()` to load a Bitmap from a file.
`handleDrawings()`: It uses `BitmapFactory.decodeStream()` to load a Bitmap from an input stream obtained from opening a file.

6. Mutable Bitmap:
- `loadFromStorage()`: It sets the `inMutable` option to `true` for the loaded Bitmap.
- `handleDrawings()`: It does not set the `inMutable` option, so the loaded Bitmaps may not be mutable.

7. Source of File Names:
- `loadFromStorage()`: It doesn't involve a list of file names; it directly specifies the file name ("drawing.png").
- `handleDrawings()`: It processes a list of file names passed as a parameter.

8. Storage Location Type:
- `loadFromStorage()`: Loads from external public storage, which may require appropriate permissions.
- `handleDrawings()`: Loads from internal storage, which does not require additional permissions but is only accessible to the app itself.

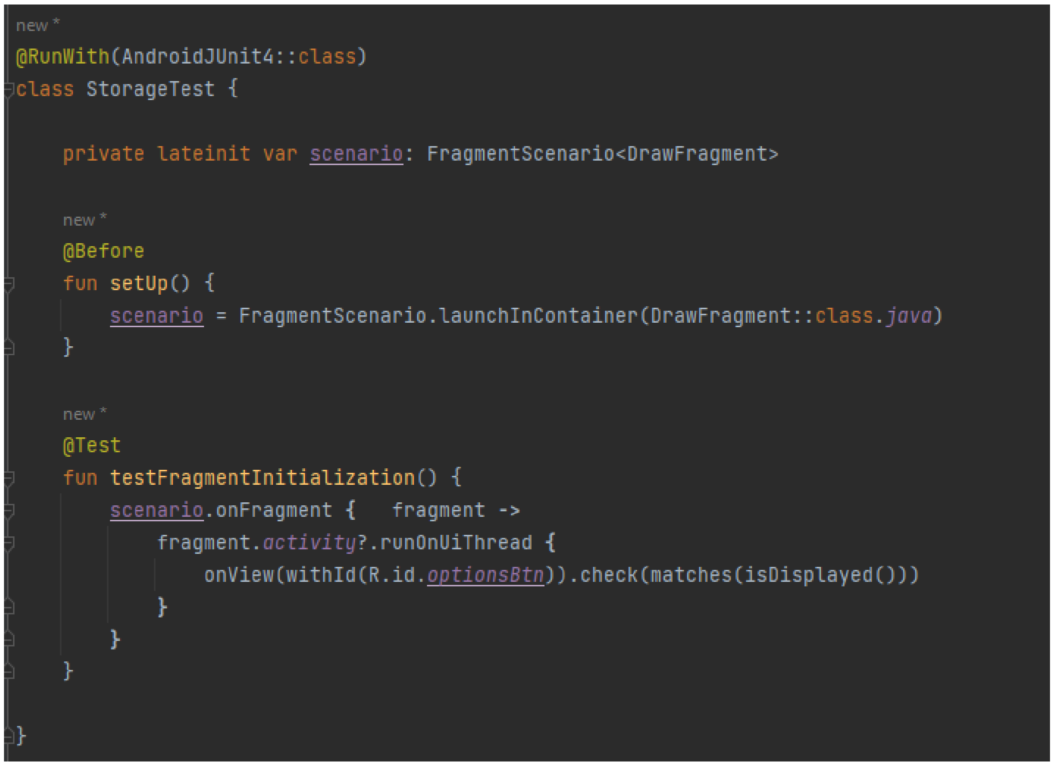
9. Use Case:
- `loadFromStorage()` is more suitable for loading a specific image file shared publicly, such as a user's drawing.
- `handleDrawings()` is more suitable for loading and handling a collection of images stored privately within the app's data directory.

In summary, these two implementations have different use cases and use different methods to load images from storage. The `loadFromStorage()` function is focused on loading a specific image from external storage, while `handleDrawings()` is designed to load and process multiple images from internal app storage. The choice between them depends on the specific requirements of your application.

3. Testing

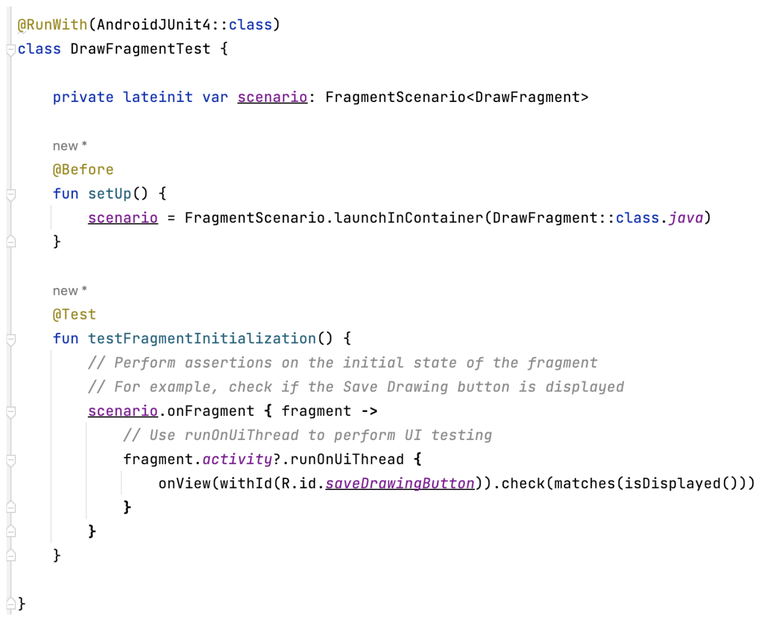
Pick part of your code that you don't think is well tested. Work with your partner to add more tests to exercise that part of your project and commit them to your repo

1. Jowie's Test:
   Since I had to fix my save button, I had to follow Zidong's test. For now, the load function is incredibly difficult for me to test. I will have to inquire more in the future.



```
new *
@RunWith(AndroidJUnit4::class)
class StorageTest {

    private lateinit var scenario: FragmentScenario<DrawFragment>

    new *
    @Before
    fun setUp() {
        scenario = FragmentScenario.launchInContainer(DrawFragment::class.java)
    }

    new *
    @Test
    fun testFragmentInitialization() {
        scenario.onFragment {   fragment ->
            fragment.activity?.runOnUiThread {
                onView(withId(R.id.optionsBtn)).check(matches(isDisplayed()))
            }
        }
    }

}
```

2. Zidong's Test:
   Test added:



```
@RunWith(AndroidJUnit4::class)
class DrawFragmentTest {

    private lateinit var scenario: FragmentScenario<DrawFragment>

    new *
    @Before
    fun setUp() {
        scenario = FragmentScenario.launchInContainer(DrawFragment::class.java)
    }

    new *
    @Test
    fun testFragmentInitialization() {
        // Perform assertions on the initial state of the fragment
        // For example, check if the Save Drawing button is displayed
        scenario.onFragment { fragment ->
            // Use runOnUiThread to perform UI testing
            fragment.activity?.runOnUiThread {
                onView(withId(R.id.saveDrawingButton)).check(matches(isDisplayed()))
            }
        }
    }

}
```